Trade-off between energy savings and execution time applying DVS to a microprocessor

Miroslav Vasić, Oscar García, Pedro Alou, Jesús A. Oliver, José A. Cobos
Universidad de Politécnica en Madrid, Centro de Electrónica Industrial, Spain

Abstract

DVS (Dynamic Voltage Scaling) is a technique used for reducing the power consumption of microprocessors. The power consumed by these circuits has a main component (dynamic power) that is proportional to the square of the supply voltage. Additionally, for every supply voltage, there is a maximum value of the clock frequency. The advantage of using DVS is that the supply voltage (and hence clock frequency) can be adjusted depending on the specific needs during execution. The DVS concept has been used in some commercial products like Transmeta's Crusoe [1], Intel Speed Step [2], AMD K6 [3], Hitachi SH4 [4], etc.
The DVS algorithm proposed in this work is based on the trade-off between the application's execution time and the energy consumed by the microprocessor. Indirectly, by controlling the execution time the consumed energy is controlled as well. Longer execution time provides less energy demanded by the CPU. The algorithm has been implemented on a platform with an Intel XScale PXA255 microprocessor and the energy saving has been calculated directly measuring currents and voltages on the platform.
Using this technique it is possible to achieve up to 50% of power savings, with 50% longer execution time.

1 Introduction

Decreasing power consumption in digital circuits is getting on importance in these days, especially for mobile and battery operated system and large data centers. The reduction of the power consumption implies more autonomy for mobile devices and improved thermal management. Moreover, low power consumption reduces the cost on packages and heat sinks and its size and increases the circuit's autonomy and reliability. DVS is a technique used for reducing the power consumption of microprocessors. The main idea is based on the fact that the major part of the power consumption is a quadratic function of the supply voltage and directly proportional to the clock frequency as shown below:

$$P \propto C V_{DD}^2 f \qquad (1)$$

where C is the equivalent capacitance of processor's gates. Naturally, there is relation between the voltage and the maximum frequency which can be used. They are related through time delay of digital circuit:

$$t_d = k \frac{V_{DD}}{(V_{DD} - V_{th})^{\gamma}} \propto \frac{1}{f_{MAX}} \qquad (2)$$

where k, V_{th}, γ are constants which depend on the CMOS technology of the circuit.
Dynamic Voltage Scaling (DVS) is a technique that offers adjustment of the voltage and the frequency depending on the task requirements during the execution time. This adjustment can be done in a way that it takes advantage of CPU's idle times (Figure 1) or by slowing down the running processes (Figure 2).
In the first case, if there is latency, that means that associated task can be executed at lower frequency and, therefore the supply voltage can be reduced, according to Figure 1. In this way, each task will be executed with the appropriate pair of supply voltage and clock frequency to reduce power consumption as much as possible without reducing performance, by using all the available time for each task.
The second solution is based on the trade-off between the execution time and the power savings. By using lower frequencies of the CPU clock, and therefore lower supply voltages, application's execution time will increase, but the energy consumption will be lower.

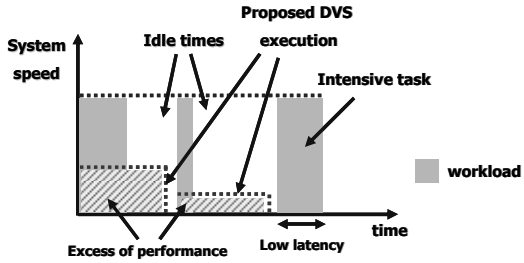

Fig.1 DVS using CPU's idle times

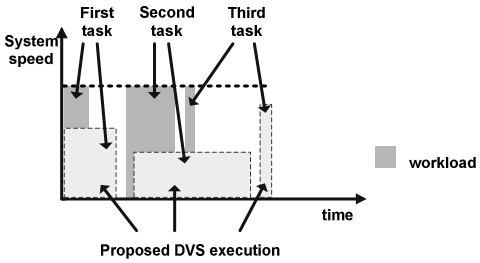

Fig.2 DVS by slowing down the running process

The DVS concept has been used in some commercial products like Transmeta's Crusoe [1], Intel's Speed Step [2], AMD K6 [3]. Thanks to this technique some

praiseworthy results were made, for example 70% of CPU energy saving for memory-bound and 15 ~ 60% for CPU bound applications [5]. In [6] this technique can prevail up to 48% of energy saving in a real-time system. For a video application in [7] it is reported 15% to 80% of energy savings, with low number of lost frames, and savings mainly depending on the complicity of the images.

The implemented DVS algorithm is based on the trade-off between execution time and power savings. By using lower frequencies of the CPU clock, and therefore lower supply voltages, the application's execution time will increase and the energy consumption will decrease. Hence, the algorithm is trying to estimate CPU activity and control the execution time of the running application, and, therefore, the consumed energy.

The algorithm has been implemented on a platform with an Intel XScale PXA255 microprocessor (has four possible CPU frequencies) and the energy saving has been calculated directly measuring currents and voltages on the platform. Using this technique it is possible to achieve up to 50% of power savings, with 50% longer execution time. The main contributions of this work are:

- Development of a closed loop algorithm that adjusts the microprocessor's frequency and voltage of the core in order to meet the desired performance.
- Development of time performance model of the system, estimated at run time.
- The adaptation of the operating system in order to facilitate and improve the measurements of the power consumption for the tested applications.

2 Proposed algorithm

The proposed algorithm is based on the decomposition of the CPU's work in workload on-chip and off-chip in order to control the execution time of the running application. The idea about the decomposition about the microprocessor's workload is presented in [5] and [8]. The workload can be presented as a sum of on-chip workload ($W_{on\text{-}chip}$) and off-chip workload ($W_{off\text{-}chip}$). On-chip workload is the number of CPU clock cycles needed to perform instructions which are executed inside the CPU only, and, on the other hand, off-chip workload is the number of external clock cycles needed to perform off-chip accesses (to fetch datafrom external memory).

Knowing the application's workload, clock and bus frequency, as well, it is possible to estimate the execution time of the running task. Hence, the problem is to estimate the workloads. Intel's family of XScale processors has a special Performance Monitoring Unit (PMU) which can monitor different CPU events as number of cache misses, number of executed instructions, number of CPU stall cycles and number of clock cycles. Using these variables it is possible to know in every moment the application's number of Stall cycles per Instruction (SPI), number of Data cache misses per Instruction (DPI) and number of Cycles per Instruction (CPI). Using the linear dependency between CPI and SPI (Figure 3), and the information about DPI, the execution time is estimated as it is explained in [5] and [8].

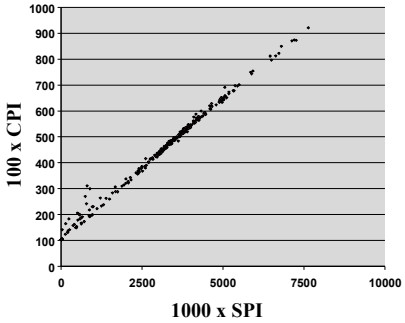

Fig. 3 The linear dependency between CPI and SPI for gzip application

The data from the PMU are taken periodically, and on the begging of each timer period, they are used to estimate the execution time. By being able to estimate application's execution time it is feasible control it by changing the CPU clock frequency, and therefore, calculate the needed frequency for each time period. In [5] and [8] power savings up to 60% are achieved.

But the solution in [5] and [8] does not have in mind the influence of the finite number of CPU frequencies. If the calculated frequency is 135MHz, for example, the applied frequency will be 100MHz, because it is the closest one (the CPU has a finite number of clock frequencies). Therefore, the active application will run slower than it is supposed, and this error is not taken into account in the next frequency calculation.

In this paper a similar algorithm is proposed, but with a feedback about the applied frequency (Figure 4) in order to compensate finite number of CPU frequencies.

In the proposed algorithm if the calculated frequency for two time intervals is, for example, 175 MHz. First is applied the frequency of 200 MHz, because it is the closest one, and then the one of 100 MHz. The algorithm would do it in the manner that the application lasts as if it was running 175 MHz all the time.

3 Proposed system model

As it was aforementioned, it is necessary to estimate the application's execution time. The estimation of the workload on-chip and the workload off-chip is done in the same way as in [6]. What is different is the frequency calculation. The microprocessor used in the tests is Intel's XScale PXA255. This processor has an internal and an external bus, and both of them are used when some data is read or written into the external memory or put in it. Having this in mind and the analysis conducted in [8] the application's execution time can be written as:

$$T = T^{ON} + T^{OFF} = \frac{W^{ON}}{f^{CPU}} + \frac{\alpha \cdot W^{OFF}}{f^{INT}} + \frac{(1-\alpha) \cdot W^{OFF}}{f^{EXT}}$$

$$T = \frac{N \cdot CPI_{on}^{avg}}{f^{CPU}} + \frac{\alpha \cdot M \cdot CPI_{off}^{avg}}{f^{INT}} + \frac{(1-\alpha) \cdot M \cdot CPI_{off}^{avg}}{f^{EXT}} \quad (3)$$

Where T^{ON} and T^{OFF} are the execution time of the on-chip workload and the off-chip workload, respectively ,and $f^{CPU}, f^{INT}, f^{EXT}$ are the CPU, the internal and the external bus frequencies, respectively. CPI_{on}^{avg} stands for the number of CPU clock cycles per an on-chip instruction and CPI_{off}^{avg} is the number of external clock cycles per an off chip instruction. N and M are the number of instructions and the number of off-chip accesses, respectively. The α is introduced as the ratio between the time which is spent due to data transfer on the internal bus and the time spent by the transfer on the external bus [8].

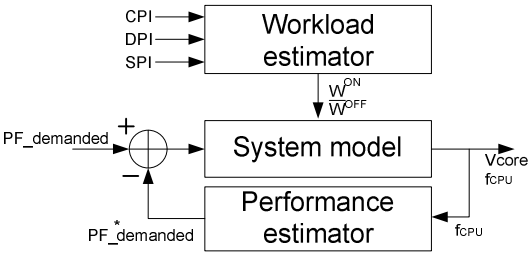

Fig. 4 The block diagram of the proposed algorithm

Table 1 shows the frequency and voltage combinations for the processor used in the tests. It is important to notice that the CPU frequency conditions the frequency of external and internal bus, in another words, they are mutually correlated.

It is necessary to define next variables, too:

$$X = \frac{f^{CPU}}{f^{EXT}} \quad Y = \frac{f^{CPU}}{f^{INT}}$$

$$\beta = \frac{T^{OFF}}{T^{ON}} = \frac{W^{OFF}}{W^{ON}} \cdot (X\alpha + Y\alpha') \ , \ \alpha' = 1 - \alpha \quad (4)$$

It can be seen that X represents the ratio of the frequency of the CPU and the frequency of the external data bus, Y stands for the ratio of the CPU's frequency and the frequency of the internal bus and β denominates the ratio of the time needed by off-chip accesses and the time spent by chip-on instructions.

Using last definitions, it can be written:

$$T^{OFF} = \frac{W^{OFF}}{f^{CPU}} \cdot (X\alpha + Y\alpha') \quad (5)$$

Due to the increased execution time it is necessary to define the application's performance loss as follows:

$$PF = \frac{T_{f^{CPU}}}{T_{f_{MAX}^{CPU}}} - 1; \quad PF \cdot T_{f_{MAX}^{CPU}} = T_{f^{CPU}} - T_{f_{MAX}^{CPU}} \quad (6)$$

where $T_{f_{MAX}^{CPU}}$ stand for the execution time at the CPU's maximum frequency, and $T_{f^{CPU}}$ is the execution time at the CPU frequency of f^{CPU}. Thus, PF shows how much the execution time of the tested application is longer than the time when the application is executed with maximal speed. For example, in the case of PF=0.2, execution time of the application with the frequency of CPU's clock of f^{CPU} is 20% longer than the time in the case when the maximal frequency is applied.

By estimating W^{ON} and W^{OFF} during one quantum period, $T_{f_{MAX}^{CPU}}$ can be estimated as well, or better said, how long this period of time would last if f_{MAX}^{CPU} were applied. Hence, equation (6) can be rewritten as:

$$PF \cdot T'_{f_{MAX}^{CPU}} = T_{f^{CPU}} - T_{f_{MAX}^{CPU}} \quad (7)$$

Doing this, the estimated "past" of the active application is separated on the left side of the equation and the "future" is on the right side.

Now it is necessary to calculate f^{CPU} for the given PF and estimated β. For a given W^{OFF} and using the data from table 1 and equations (4) and (5), as well, it can be written:

$$T^{OFF}_{f_{MAX}^{CPU}} = T^{OFF} \frac{f^{CPU}}{f_{MAX}^{CPU}} \frac{4\cdot\alpha + 2\cdot\alpha'}{X\cdot\alpha + Y\cdot\alpha'} = \beta \frac{W^{ON}}{f^{CPU}} \frac{f^{CPU}}{f_{MAX}^{CPU}} \frac{4\cdot\alpha + 2\cdot\alpha'}{X\cdot\alpha + Y\cdot\alpha'} \quad (8)$$

where T^{OFF} is the off-chip time of the application when f^{CPU} is used as a core's clock frequency and X and Y are corresponding ratios of internal and external bus frequencies and the core's frequency, as earlier defined.

Using equations (3) and (8), it is obtained:

$$T'_{f_{MAX}^{CPU}} = \frac{W^{ON}}{f_{MAX}^{CPU}} + \beta \frac{W^{ON}}{f_{MAX}^{CPU}} \frac{4\alpha + 2\alpha'}{X'\alpha + Y'\alpha'} = \frac{W^{ON}}{f_{MAX}^{CPU}} (1 + \beta \frac{4\alpha + 2\alpha'}{X'\alpha + Y'\alpha'}) \quad (9)$$

For example, let us assume that the estimated values of β and $T_{f_{MAX}^{CPU}}^{ON}$ are 4.56 and 1.8ms respectively and that in the time quantum of 10ms (when the statistics data were collected and the estimation was done) the CPU frequency was 300MHz and that α of the tested application is 0.9. Having in mind these information and the data from the table 1, $T'_{f_{MAX}^{CPU}}$ is estimated, using equations (3) and (9):

$$T'_{f_{max}^{CPU}} = 1.8ms \cdot \frac{300MHz}{400MHz} \cdot (1 + 4.56 \cdot \frac{4\cdot 0.9 + 2\cdot 0.1}{3\cdot 0.9 + 3\cdot 0.1}) = 9.1476ms$$

This means that the workload that was processed in 10ms with 300MHz system clock would have been processed in 9.1476ms if the frequency of 400MHz had been used. Multiplying this value with demanded time performance loss we obtain the amount of time for which we need to slow down the application in the next time quantum. If the PF is 40% that means that we need to choose a frequency that will process the same workload 3,659ms slower comparing with the time needed when the maximal frequency of the CPU clock is applied.

Next step is to calculate the needed frequency. Developing the right side of (6) and using (9):

$$PF \cdot \frac{W^{ON}}{f_{MAX}^{CPU}} \cdot (1 + \beta \cdot \frac{4\alpha + 2\alpha'}{X'\alpha + Y'\alpha'}) =$$

$$= \frac{W^{ON}}{f^{CPU}} - \frac{W^{ON}}{f_{MAX}^{CPU}} + \beta \cdot \frac{W^{ON}}{f^{CPU}} - \beta \cdot \frac{W^{ON}}{f_{MAX}^{CPU}} \frac{4\alpha + 2\alpha'}{X\alpha + Y\alpha'}$$

$$f^{CPU} = \frac{f_{MAX}^{CPU}(1+\beta)}{\mathrm{PF} + 1 + \beta(4\alpha + 2\alpha')(\frac{PF}{X'\alpha + Y'\alpha'} + \frac{1}{X\alpha + Y\alpha'})} \quad (10)$$

The final equation, that expresses f^{CPU} needed for the given PF and estimated β, clearly shows the influence of internal and external bus on the frequency that should be elected. When the nearest frequency is chosen, the error in time performance is recalculated using equation (10), and that information is used as it was shown in Figure 4.

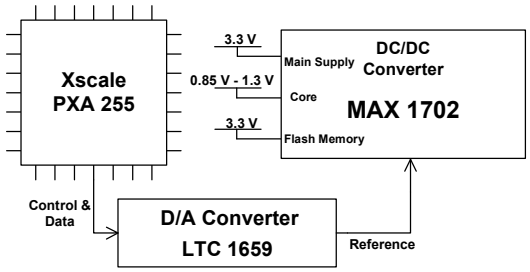

Fig. 5 The block scheme of the system

4 Hardware setup

Figure 5 shows the hardware setup used for the tests. The microprocessor can operate from 100MHz to 400MHz with a corresponding core supply voltage from 0.85V to 1.3V. This microprocessor can change the working frequency by changing the value of Core Clock Configuration Register (CCCR) [9]. The supply voltage can be changed by setting the appropriate reference to the specialized chip MAX1702. The chip contains three DC-DC converters that generate output voltages for the core, flash memory and I/O pins. The voltage reference is set by the microprocessor through the D/A converter, LTC1659. The block diagram of this part of the system is shown in Figure 5. The operating system is Linux 2.6.13 with RTCore extension. The DVS algorithm is implemented as a real-time module of RTCore and acts as a periodic task. The used operating system is preemptive, i.e. the process with the highest level of priority is the active one. Thus, the tested application can be preempted by any process with higher priority. In order to facilitate the measurement of the power consumption and to be sure that the measured power is used by the tested application, a little adaptation of the operating system was done. The application that is tested is recognized directly in the scheduler of the operating system. When the scheduler marks this application as the active, one of the output pins is set to logical 1. Therefore, by measuring the power consumed by the CPU and the time intervals when this signal is active the execution time and the energy consumed by the tested application can be determined. The measurement of the energy consumed by the DVS algorithm is done in the similar way. Using this method it is not necessary to know the priority level of tested application in order to measure its energy consumption, as the control signal shows us when the tested application is active.

5 Experimental results

The proposed algorithm has been tested with three different applications: gzip (compression of files), bfish (file encoding) and cjpeg (compression of photos). The time interval used to collect CPU data and estimate its workload in all tests was 20ms. Figure 6 shows the actual performance loss of the system. It can be seen that is very close to the values that are demanded, all values are in ±5% of $PF_{DEMANDED}$. Figure 7 presents the energy saving and it can be noticed that they are asymptotically approaching certain value. The reason for this is that the energy savings cannot be higher than in the case of minimal CPU frequency. By increasing the $PF_{DEMANDED}$, the average CPU frequency is getting closer to 100 MHz, so that energy savings are drawing near the maximum savings. Figure 8 shows that without closing the loop in the system some values of Performance Loss cannot be accomplished correctly, for instance when the demanded PF is 50%, the difference between the compensated algorithm and the algorithm without compensation is about 10%.

The time and the energy spent during frequency/voltage transitions should be negligible. Figures 9 and 10 show the execution time and the consumed energy of the proposed DVS algorithm. The presented values are calculated regarding the time/energy of the tested application at the maximum speed. The maximum values are less than 3% of application's performance and energy.

Each transition time is composed of the time needed for the voltage change and the time spent by PLL to lock to the new frequency. By measuring these times it is determined that, approximately, 80% of the time needed by one transitions is spent by PLL. In order to see the influence of the power supply's dynamics we measured time and energy needed by the algorithm for two cases of supply's slew rate. Figure 11 shows that because of the strong influence of PLL the dynamic of the power supply does not have great influence on the execution time of the algorithm. For this test the time interval of frequency/voltage changes was 5ms.

As it was aforementioned, the time needed for off chip activities is estimated using a constant α. The value of the constant is evaluated by conducting a series of experiment in order to find the best value that would suite to whole range of PF_{WANTED}. Hence, before applying the algorithm it is necessary to characterize the application and find the optimum α. Obviously, this is a drawback of the proposed algorithm. Even more, we found out that for the same application we need to adjust the value of α depending on the type of file that application uses as the object of its algorithm. For example, compression of a file filled with ASCII code (txt files) needs α of 0.95, and in the case of the same compression, but this time of an Acrobat Reader file (pdf), we could not find optimum value of α for all the range of PF_{WANTED}. Similar problem we found out trying to compress a black and white photo to jpg format.

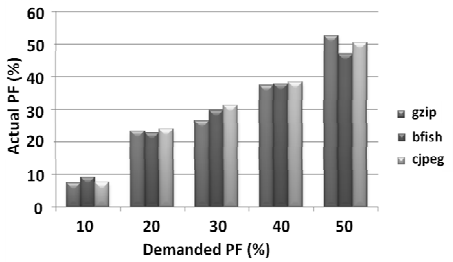

Fig. 6 Actual vs. demanded time performance loss

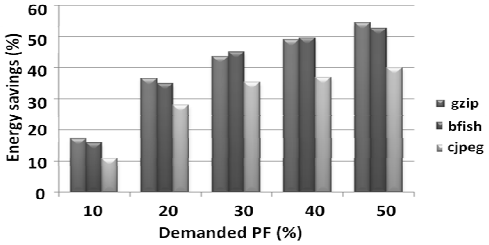

Fig. 7 Energy savings vs. demanded time performance loss

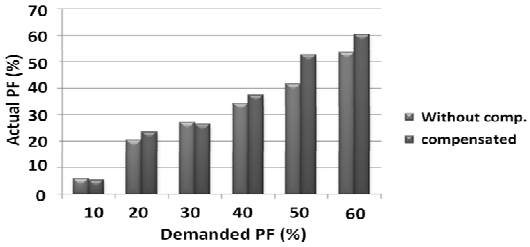

Fig. 8 Compensated algorithm vs. algorithm without compensation

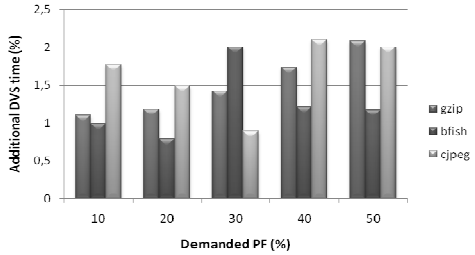

Fig. 9 Additional DVS time vs. demanded time performance loss

The main cause of this problem can be in the statistics of the tested application. If we compare the statistics for gzip application in the case when we want to compress a txt file and when we want to compress a pdf file, we can see a significant difference between these two processes. In figure 12 CPI vs. SPI diagram for gzip compressing a pdf file is presented. After a series of tests we came with assumption that the algorithm did not model the time needed to read a source file, and later to write the results to an output file, but it is yet to be proved.

6 Conclusions

A new energy savings algorithm for DVS applications is proposed. The goal is to improve the energy efficiency by demanding some acceptable performance loss. According to the experimental results, it is possible to save up to 50% of the CPU energy with 50% performance loss. The extra time needed for the voltage/frequency transitions and the energy consumed by those transitions do not exceed 3% and 2% of the time and the energy of the tested application, respectively. The analysis of the extra time needed for DVS transitions has been performed, showing that it does not need a voltage supply with fast transitions, due to long duration of frequency changes. The algorithm is strongly dependent and very sensitive on good estimation of CPU workload, what is very difficult to obtain in some cases.

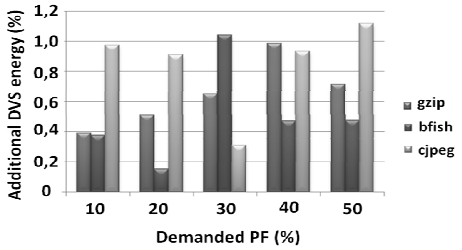

Fig. 10 Additional DVS energy vs. demanded time performance loss

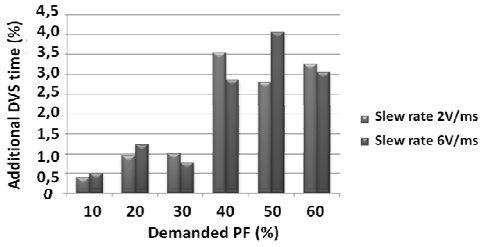

Fig. 11 Additional DVS time for different slew rates of the power supply

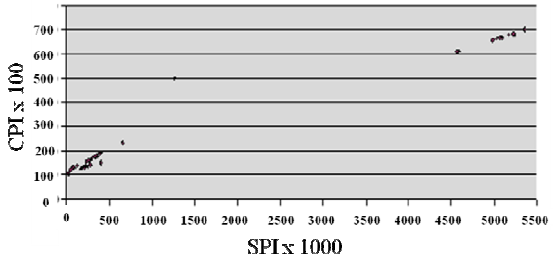

Fig. 12 Dependency between CPI and SPI when the algorithm does not work very well

7 Literature

[1] Transmeta's Design guides and Datasheets

[2] Enhanced Intel SpeedStep, White paper, March 2004.

[3] Mobile AMD-K6 Processor Power Supply Design, Application Note

[4] Kawaguchi, H.;Shin Y.;Sakurai T.: uITRON-LP: Power-Conscious Real-Time OS Based on Cooperative Voltage Scaling for Multimedia Applications. IEEE trans. on multimedia, Feb. 2005.

[5] Choi, K.; Soma, R.; Pedram, M.: Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Tradeoff Based on the Ratio of Off-Chip Access to On-Chip Computation Times. IEEE trans. on computer-aided design of integrated circuits and systems, vol. 24, No. 1, Jan. 2005

[6] P. Pillai, K. G. Shin, " Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems" Proc. of the 18th ACM Symp. on Operating Systems Principles, 2001

[7] D. Son, C. Yu, H. Kim, "Dynamic Voltage Scaling on MPEG Decoding" International Conference of Parallel and Distributed System (ICPADS), June 2001

[8] Choi, K.; Soma, R.; Pedram, M: Dynamic Voltage and Frequency Scaling based on workload Decomposition. Proceedings of the 2004 International Symposium on Low Power Electronics and Design, ISPLED, 2004

[9] Intel PXA255 Processor, Developer's manual, 2004.